“PTU COMPLAINT MANAGEMENT SYSTEM”

Problem Definition:

At Puducherry Technical University (PTU), submitting complaints involves a cumbersome process. Students, professors, and supporting staff must identify the appropriate authority, send emails manually, and follow up individually to check the status of their complaints. Gaining community support for issues often requires additional manual communication. Submitting anonymous complaints is also inconvenient, requiring a separate email ID. This traditional method creates confusion and inefficiency for all parties.

To address these challenges, I developed a PTU Complaint Management System. This MERN stack web application streamlines the complaint submission and resolution process. It allows users (students, professors, supporting staff) to register, log in, view existing complaints, express support via likes/dislikes, and raise new complaints, either publicly or anonymously. Admins (authorized professors with a passkey) can monitor all complaints, update statuses, and provide resolutions efficiently through a secure dashboard

Architecture:

The system is built using the MERN stack with the following technologies:

- **Frontend**: React.js
- **Backend**: Node.js with Express.js
- **Database**: MySQL
- **API Support**: RESTful API

SOFTWARE REQUIREMENTS SPECIFICATION (SRS)

1. Introduction

1.1 Purpose

The purpose of this system is to provide a centralized web-based complaint management platform for the PTU campus community. It enables students, professors, and supporting staff to raise, track, and support complaints easily while allowing admin authorities to manage and resolve them efficiently. The system reduces dependency on email-based complaints and improves transparency and accessibility.

1.2 Scope

This MERN-stack based complaint management system includes:

- **Modules**: User registration, complaint creation, complaint tracking, like/dislike support system, role-based access control.
- **Role-based Access**:
- Users (students, professors, supporting staffs): Can register, log in, raise complaints (public or private), support existing complaints, view status and solutions.

- Admins (professors/authorized officials): Can view all complaints, update status and solutions.
- **Frontend**: React.js
- **Backend**: Node.js with Express.js
- **API**: REST API support
- **Database**: MySQL

1.3 Intended Audience

- PTU Students, Professors, and Supporting Staff
- PTU Admin Authorities
- Developers and Testers working on the project

1.4 Assumptions & Dependencies

- MySQL and Node.js are installed and configured
- Admins are provided with valid passkeys for account creation
- The system is currently deployed on localhost
- No third-party authentication or visualization tools are integrated
- Security features like JWT or HTTPS are not yet implemented but planned for future updates

2. Overall Description

2.1 Product Perspective

This project is a standalone web application that simplifies complaint submission and management at PTU. It streamlines the previously manual and email-based complaint process, encouraging transparency and timely resolution. The architecture is scalable for adding more modules like comments or feedback in the future.

2.2 User Classes and Characteristics

Role	Description
User	Students, professors, and supporting staff. Can register, raise complaints, view public complaints, like/dislike, and check status and solutions.
Admin	PTU authority with passkey access. Can manage complaints, view all types (public/private), and update statuses and solutions.

2.3 Operating Environment

- Modern web browser with JavaScript enabled (for frontend React)
- Backend server running Node.js + Express
- MySQL Database server
- Localhost deployment environment
- The following sections describe the functional and non-functional requirements of the system.

3. System Features (Functional Requirements)

- **User Authentication & Authorization**
  - User registration with role as Student, Professor, or Supporting Staff, capturing relevant details based on role
  - Admin registration with role as Admin

- Login and logout functionality
- Role-based access control for users and admins
- Session handling via local state (no JWT or cookies currently)

❖ **Complaint Creation**
- Users can create new complaints
- Options while creating complaint:
  Public (includes name and details)
  Private (submitted anonymously)
- Expose setting:
  Expose = Yes (visible to others)
  Expose = No (only admin can see)
- Full complaint body field for detailed description

❖ **Complaint Viewing**
- Users can view all public complaints with Expose = Yes
- Like/Dislike functionality to show support
- Complaints display current status and solution (if updated)

❖ **Complaint Tracking & History**
- Users can view a list of complaints they've submitted
- Track complaint status and see updates from admin

❖ **Admin Complaint Management**
- Admin can view all complaints, regardless of visibility settings
- Admin can update:
  Status: Pending, In Progress, Resolved
  Solution: Add action taken or comments
- Admin dashboard for streamlined complaint management

❖ **Privacy Options**
- Users can choose to submit complaints anonymously
- Private complaints exclude issuer identity
- Expose control ensures selective visibility

❖ **Complaint Support System**
- Users can support complaints via Like/Dislike
- Like counts help prioritize high-impact issues

❖ **User Profile Management**
- Users can view and update their profile
- Access full complaint history via My Complaints section

❖ **Scalability (Planned Features)**
- Add commenting system on complaints
- Notification system for status updates
- Secure login using JWT and role middleware
- Cloud deployment using platforms like Vercel or Railway

4. Non-Functional Requirements

4.1 Performance

- Supports 50+ concurrent users on localhost setup
- REST API calls return responses typically under 1 second in local environment
- Efficient complaint listing and like/dislike operations

4.2 Security

- Currently no JWT or OAuth implemented (planned for future)
- Admin access controlled through secure passkey system
- Basic role-based access control (User vs Admin) handled at backend

4.3 Usability

- Clean, minimal React frontend
- Main navigation options:
  - Home
  - New Complaint
  - My Complaints
  - Profile
- Inside the Home page, sub-navigation tabs include:
  - All Complaints
  - Students' Complaints
  - Professors' Complaints
  - Supporting Staffs' Complaints
  - Private Complaints
- Complaint form supports public/private and expose yes/no options
- Real-time feedback on form errors and submission status

4.4 Scalability

- Easily extendable to include features like:
  - Public/private commenting system
  - Complaint filtering and sorting
  - Admin-level analytics on complaints raised
- Modular design allows easy integration of new modules such as feedback system or suggestion box

5. External Interfaces

5.1 User Interface

- Built using **React.js**
- Role-based access: **Admin** and **User** views
- Clean dashboard for complaint viewing and management
- Tabs for filtering complaints: All, Students, Professors, Supporting Staffs, Private
- Real-time feedback during form interactions (errors, success messages)

5.2 Hardware Interfaces

- No specific hardware dependencies
- Compatible with standard devices: desktop, laptop, mobile

5.3 Software Interfaces

- Frontend: React.js
- Backend: Node.js with Express.js
- Database: MySQL via Sequelize ORM
- APIs: RESTful APIs for all operations
- Authentication: Secure passkey system (JWT planned for future)

5.4 Communication Interfaces

- All data communication over **HTTP (local)** / **HTTPS (when deployed)**
- Data transmitted in **JSON** format

PROJECT STRUCTURE

BACKEND

Folder structure:

```
BACKEND
|
|_ node_modules
|
|_ src
|    |
|    |_ config
|    |   |_ db.config.js
|    |
|    |_ controllers
|    |   |_ admins.controller.js
|    |   |_ privateComplaints.controller.js
|    |   |_ professors.controller.js
|    |   |_ publicComplaints.controller.js
|    |   |_ students.controller.js
|    |   |_ supportingStaffs.controller.js
|    |
|    |_ middleware
|    |
|    |_ models
|    |   |_ admins.model.js
|    |   |_ privateComplaints.model.js
|    |   |_ professors.model.js
|    |   |_ publicComplaints.model.js
|    |   |_ students.model.js
|    |   |_ supportingStaffs.model.js
|    |
|    |_ repositories
|    |   |_ admins.repository.js
|    |   |_ privateComplaints.repository.js
|    |   |_ professors.repository.js
|    |   |_ publicComplaints.repository.js
|    |   |_ students.repository.js
|    |   |_ supportingStaffs.repository.js
|    |
|    |_ routes
|    |   |_ admins.routes.js
|    |   |_ privateComplaints.routes.js
|    |   |_ professors.routes.js
|    |   |_ publicComplaints.routes.js
|    |   |_ students.routes.js
|    |   |_ supportingStaffs.routes.js
|    |
|    |_ services
|    |   |_ admins.service.js
|    |   |_ privateComplaints.service.js
|    |   |_ professors.service.js
|    |   |_ publicComplaints.service.js
|    |   |_ students.service.js
|    |   |_ supportingStaffs.service.js
|    |
|    |_ app.js
|    |_ server.js
|
|_.env
|
|_package-lock.json
|
|_package.json
```

node_modules/ – Node Package Dependencies

Contains all npm packages and dependencies installed for the backend project.

src/ – Application Source Code Root

This is the main working directory for backend logic, divided by responsibility layers.

config/ – Configuration Layer

Handles external configurations like database connections.

- db.config.js – Contains MySQL database connection details, Sequelize settings, and credentials.

models/ – Data Schema Layer

Defines Sequelize models representing database tables/entities.

- admins.model.js – Defines the schema for admin users with required fields.
- privateComplaints.model.js – Represents anonymous complaints data.
- professors.model.js – Represents professor user accounts and their details.
- publicComplaints.model.js – Schema for public complaint submissions.
- students.model.js – Defines schema for student users.
- supportingStaffs.model.js – Schema for supporting staff users.

controllers/ – Request Handling Layer

Manages HTTP requests, calls appropriate services, and returns responses.

- admins.controller.js – Handles admin registration, login, and all other admin actions.
- privateComplaints.controller.js – Manages private complaint submissions and access.
- professors.controller.js – Handles professor-specific operations.
- publicComplaints.controller.js – For creating and viewing public complaints.
- students.controller.js – Handles student-specific operations.
- supportingStaffs.controller.js – Handles supporting staffs-specific operations.

repositories/ – Data Access Layer

Implements Sequelize-based database operations for each model, abstracting raw queries.

- admins.repository.js – DB access methods for admin functionalities.
- privateComplaints.repository.js – For inserting/fetching private complaints.
- professors.repository.js – Access layer for professor-related DB actions
- publicComplaints.repository.js – Manages public complaint queries.
- students.repository.js – Access layer for student-related DB actions.
- supportingStaffs.repository.js – Access layer for supporting staff-related operations.

routes/ – API Routes

Maps HTTP request endpoints to controller functions.

- admins.routes.js – Routes for admin registration, login, and complaint management.
- privateComplaints.routes.js – API routes for private complaint operations.
- professors.routes.js – Routes specific to professor user data.
- publicComplaints.routes.js – Endpoints to submit/view public complaints.
- students.routes.js – Student-specific APIs (register, view profile, etc.).

- supportingStaffs.routes.js – Routes for supporting staff operations.

🧠 services/ – Business Logic Layer

Contains core logic functions invoked by controllers before interacting with the database.

- admins.service.js – Logic for admin creation, login validation, complaint review.
- privateComplaints.service.js – Handles logic for creating private complaints.
- professors.service.js – Business logic for managing professor data.
- publicComplaints.service.js – Handles complaint submission and visibility rules.
- students.service.js – Registration, update, and profile view logic for students.
- supportingStaffs.service.js – Handles supporting staff registration and interaction flow.

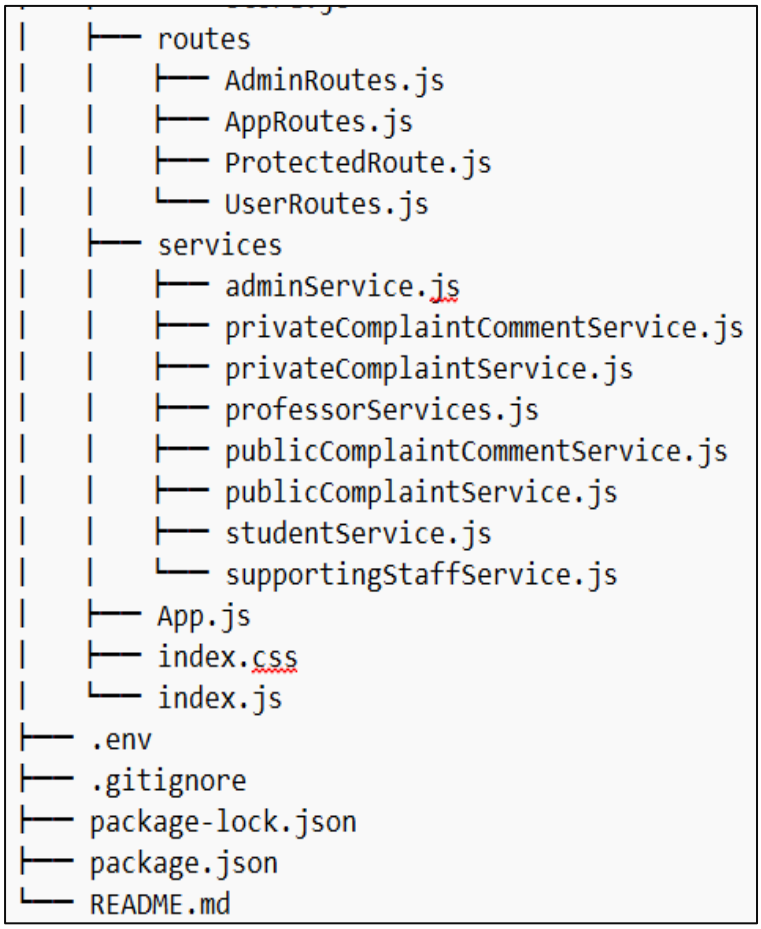
🛠️ Root-Level Files

- app.js – Initializes Express app, applies middleware, routes, and error handlers.
- server.js – Starts the Node.js server and connects to the database.

FRONTEND

Folder Structure:

```
FRONTEND
├── node_modules
├── public
├── src
|   ├── components
|   |   ├── AdminComplaintFeed.js
|   |   ├── AdminPrivateComplaintBlock.js
|   |   ├── AdminPublicComplaintBlock.js
|   |   ├── AdminRegister.js
|   |   ├── ComplaintFeed.js
|   |   ├── Footer.js
|   |   ├── Header.js
|   |   ├── Navbar.js
|   |   ├── PrivateComplaintBlock.js
|   |   ├── ProfessorRegister.js
|   |   ├── PublicComplaintBlock.js
|   |   ├── StudentRegister.js
|   |   └── SupportingStaffRegister.js
|   |
|   |── css
|   |   ├── About.css
|   |   ├── App.css
|   |   ├── ComplaintBlock.css
|   |   ├── ComplaintFeed.css
|   |   ├── FlashPage.css
|   |   ├── Footer.css
|   |   ├── Header.css
|   |   ├── Home.css
|   |   ├── Login.css
|   |   ├── Navbar.css
|   |   ├── NewComplaint.css
|   |   ├── Profile.css
|   |   ├── Register.css
|   |   ├── RegisterPage.css
|   |   ├── ViewComplaint.css
|   |   └── YourComplaint.css
|   |
|   ├── images
|   |   └── logo.webp
|   ├── layout
|   |   ├── AuthLayout.js
|   |   └── MainLayout.js
|   ├── pages
|   |   ├── About.js
|   |   ├── AdminHome.js
|   |   ├── AdminProfile.js
|   |   ├── FlashPage.js
|   |   ├── Home.js
|   |   ├── Login.js
|   |   ├── NewComplaint.js
|   |   ├── Profile.js
|   |   ├── Register.js
|   |   ├── ViewComplaint.js
|   |   └── YourComplaints.js
|   ├── redux
|   |   ├── actions
|   |   |   ├── adminsActions.js
|   |   |   ├── privateComplaintActions.js
|   |   |   ├── professorsAction.js
|   |   |   ├── publicComplaintActions.js
|   |   |   ├── studentsActions.js
|   |   |   └── supportingStaffsActions.js
|   |   ├── reducers
|   |   |   ├── adminReducer.js
|   |   |   ├── privateComplaintsReducer.js
|   |   |   ├── professorsReducer.js
|   |   |   ├── publicReducer.js
|   |   |   ├── rootReducer.js
|   |   |   ├── studentReducer.js
|   |   |   └── supportingStaffsReducer.js
|   |   └── store.js
```



public/ – Public Assets

Contains files served directly to the browser and accessible without authentication.

- index.html – Main HTML file with the root div where the React app mounts.
- favicon.ico – Icon shown in the browser tab.
- manifest.json, robots.txt – *(Optional files)* for PWA setup and crawler directives.

src/ – Main Application Source Code

All the React components, pages, logic, and routing configuration reside here.

- **App.js**
  Main React component where routing and layout wrappers are initialized.
- **index.js**
  Bootstraps the React app into the DOM and wraps the application in global providers (e.g., Redux).

components/ – Reusable UI Components

Includes individual components reused across different pages and views.

- Navbar.js – Top navigation bar with role-based links.
- Footer.js, Header.js – UI wrappers for top and bottom layouts.
- ComplaintFeed.js – Lists all visible complaints.
- PublicComplaintBlock.js, PrivateComplaintBlock.js – Render individual complaint cards.

- AdminPublicComplaintBlock.js, AdminPrivateComplaintBlock.js, AdminComplaintFeed.js – Admin-specific views for complaint moderation.
- StudentRegister.js, ProfessorRegister.js, SupportingStaffRegister.js, AdminRegister.js – Registration forms based on user roles.

components/css/ – Component-specific styles

CSS modules to style each component individually.

- Styles include: Navbar.css, ComplaintFeed.css, NewComplaint.css, Profile.css, Login.css, Home.css, ViewComplaint.css, and more.

pages/ – Route-Linked Views

- Each file represents a full-page screen, typically mapped via routing.
- Home.js – Displays complaints and navigation tabs.
- FlashPage.js – Temporary landing page.
- Login.js – Handles user/admin login.
- Register.js – Registration form redirection for all roles.
- NewComplaint.js – Form to raise a new complaint.
- ViewComplaint.js, YourComplaints.js – Detailed views for user's complaints.
- Profile.js, AdminProfile.js – Profile screens for users and admins.
- AdminHome.js, About.js – Admin dashboard and about info page.

redux/ – Global State Management (Redux Toolkit)

Handles application-wide state including users, complaints, etc.

actions/ – Dispatchable Functions

- adminsActions.js, studentsActions.js, supportingStaffsActions.js – CRUD and async actions.
- publicComplaintActions.js, privateComplaintActions.js, professorsAction.js – Complaint-related logic.

reducers/ – State Update Handlers

- Manages corresponding slices of the Redux state tree.
- adminReducer.js, studentReducer.js, publicReducer.js, rootReducer.js, etc.

store.js

Combines all reducers and initializes the global Redux store.

routes/ – React Router Configuration

Defines protected and role-specific route guards.

- AdminRoutes.js, UserRoutes.js – Routes accessible by specific roles.
- AppRoutes.js – Main routing setup used inside App.js.
- ProtectedRoute.js – Auth check wrapper to protect pages.

services/ – API Service Layer

Centralized service files for making API calls using Axios or Fetch.

- adminService.js, studentService.js, professorServices.js

- privateComplaintService.js, publicComplaintService.js, privateComplaintCommentService.js, etc.

🖼 images/ – Static Media

Contains app images, logos, and icons.

- logo.webp – University logo or application branding.

🗀 Root-Level Files

- .env – Environment configuration (API URLs, secrets).
- .gitignore – Git version control exclusions.
- package.json – NPM dependencies and script declarations.
- package-lock.json – Dependency lock file.
- README.md – Project overview and setup guide.
- index.css – Global styles for the app.

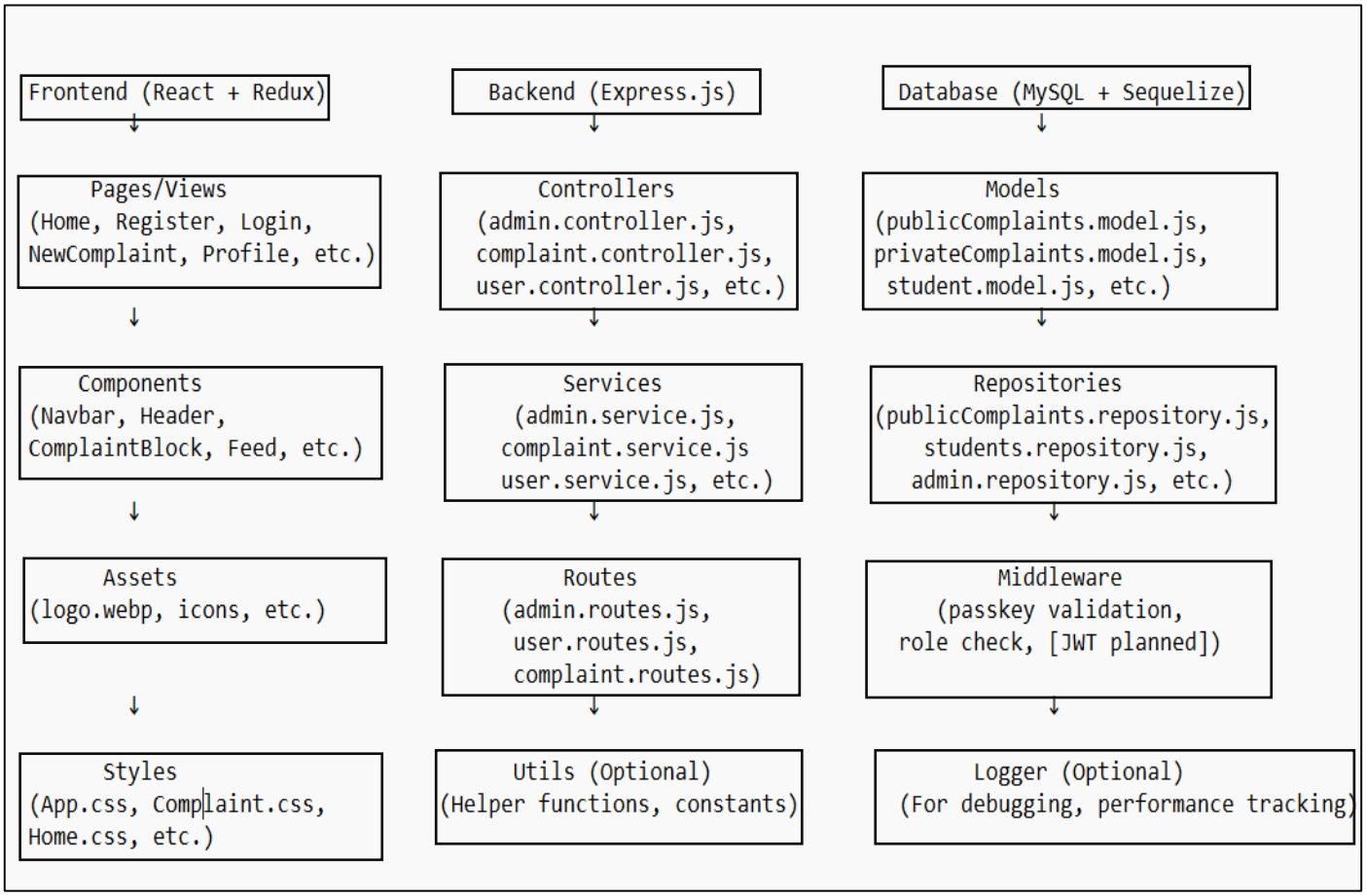
🧩 Folder Structure Summary

This modular React structure supports:

- **Separation of concerns**: Components, pages, services, redux, and routing are cleanly isolated.
- **Scalability**: Easy to plug in more user roles, services, or route types.
- **Maintainability**: CSS modules per component make styling modular and clean.
- **Extendibility**: You can easily add features like notifications, dashboards, analytics, or charts.

CLEAN ARCHITECTURE BLOCK DIAGRAM

Data Flow Diagram:



Purpose:

To show how data moves between **users** (through the frontend), the **backend API** (routes, services, controllers), and the **database** (via Sequelize ORM with MySQL) in your complaint management system.

Components

Actors:

- **Public User (Student, Professor, Supporting Staff)**:
  Registers, logs in, raises public/private complaints, views complaint status, supports complaints via like/dislike.
- **Admin (Official Authority)**:
  Logs in using passkey, views and filters all complaints, updates complaint status and solutions.

Processes:

- **User Login**
  - Login page (Login.js)
  - authReducer & authActions dispatch credentials

- POST /api/students/login or /api/professors/login or /api/supportingstaffs/login
- Backend validates credentials
- If valid, session state is stored locally (JWT planned in future).

- **User Raises a Complaint**
  - NewComplaint.js submits form
  - POST /api/publicComplaints or /api/privateComplaints
  - Handled by publicComplaints.controller.js
  - Passed to publicComplaints.service.js
  - Saved via publicComplaints.repository.js using Sequelize to MySQL.

- **Admin Updates Complaint**
  - Admin dashboard (AdminComplaintFeed.js)
  - PUT request to /api/publicComplaints/:id
  - publicComplaints.controller.js → service → repository
  - Updates solution/status in the database.

- **Complaint Fetching (Public View)**
  - ComplaintFeed.js or ComplaintBlock.js
  - GET /api/publicComplaints?expose=true
  - Filters only visible complaints
  - JSON data returned and rendered as cards in the UI.

- **Like/Dislike Support**
  - PublicComplaintBlock.js → Click like
  - PUT request to /api/publicComplaints/like/:id
  - Backend updates like count
  - UI re-renders updated count.

Explanation:

[User (Student / Professor / Supporting Staff) / Admin] <----> [Frontend (React + Redux)]

↓

[REST API Layer (Express.js, Routes)]

↓

[Controllers → Services (Business Logic) → Repositories]

↓

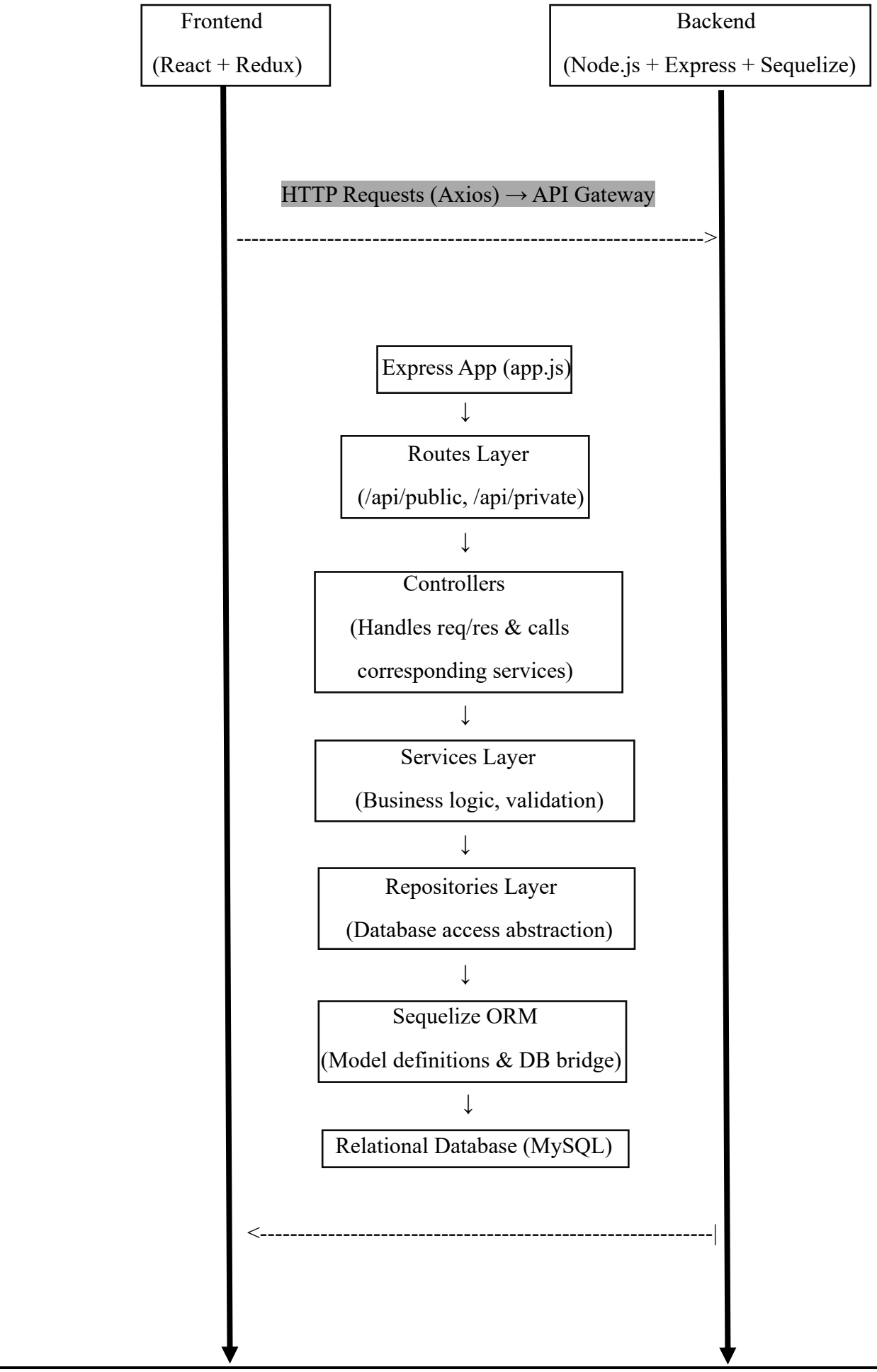
[Sequelize ORM Models (MySQL Database)]

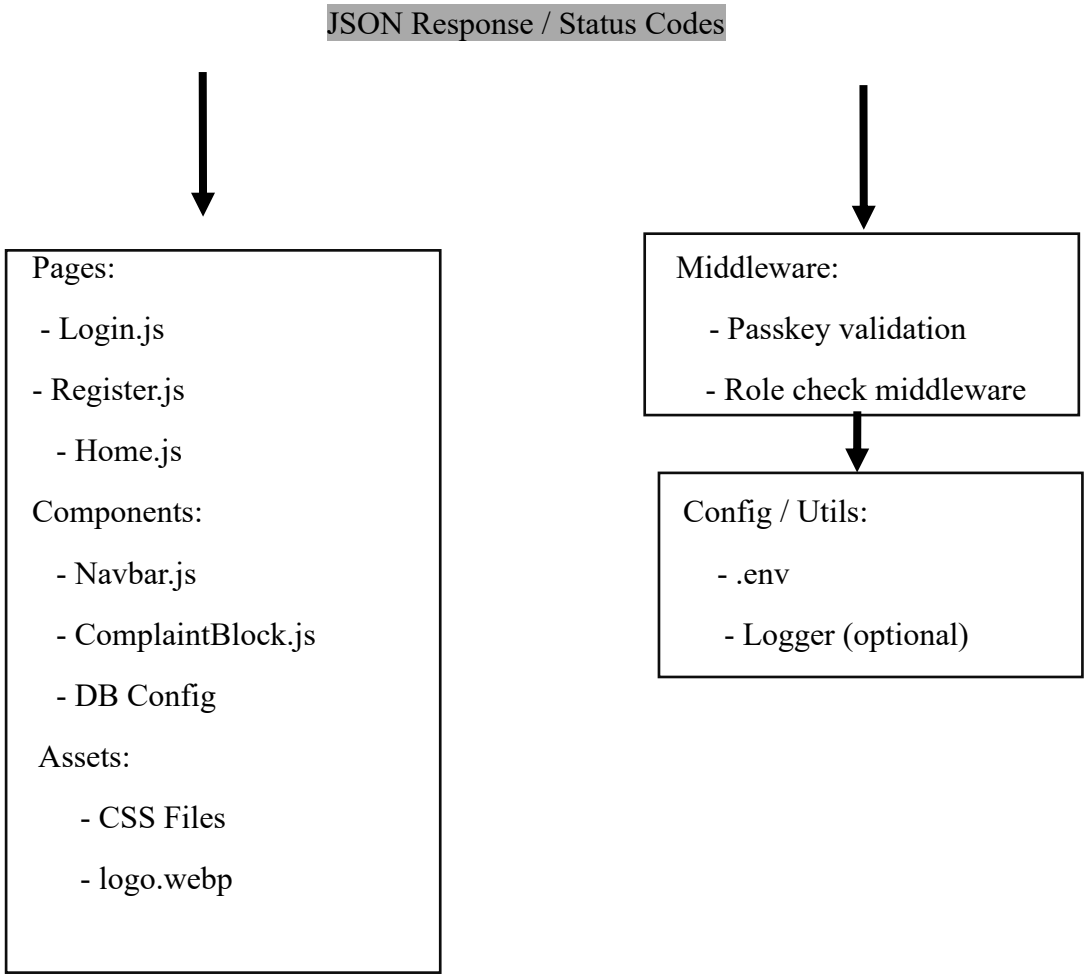
- All interactions begin in the **React frontend**, where Redux dispatches actions (e.g., login, submit, update).
- These actions hit **Express.js routes**, which forward the logic to **controllers** and **services**.
- Services handle validation and call **repositories**, which directly communicate with **Sequelize models**.
- Data is stored or fetched from the **MySQL database** based on request type.
- Admins and users have **role-based** access (currently handled with passkey; JWT planned for secure role middleware).

🔁 Data Flow Summary

Layer	Description
Frontend	React pages/components (e.g., Register, Home, NewComplaint) trigger Redux actions.
Redux/API	actions send HTTP requests to backend Express API.
Routes	Define endpoint paths and delegate logic to appropriate controllers.
Controllers	Receive request, call services, and return response.
Services	Business logic like form validation, update rules, access checks.
Repositories	Perform raw DB queries using Sequelize.
Models	Define table schemas (e.g., students.model.js, publicComplaints.model.js).

PTU complaint Management System — Architecture Diagram (Text-Based)





🎯 **Purpose:**

To represent how the **PTU Complaint Management System** is logically divided into clean, testable layers that separate responsibilities and support secure, scalable development.

👥 **Actors:**

- **User (Student / Professor / Supporting Staff):**
  Can register, log in, raise complaints (public/private), view complaint status, and support complaints.
- **Admin (Official Authority):**
  Can view all complaints, update their status/solution, and manage the complaint flow across all categories.

🏵 **Processes:**

- **User views public complaints:**
  UI → API → Controller → Service → Repository → Database → JSON Response.
- **User submits a new complaint (public/private):**
  UI form (React) → Redux Action → Express Route → Controller → Service → Repository → Data saved via Sequelize in MySQL.

- **Admin updates complaint status or solution:**
  Admin Dashboard → PUT Request → API Route → Controller → Business Logic → DB update via Sequelize → Response.

🔁 **Flow Summary:**

- **Frontend (React + Redux):**
  React handles UI + user input → Redux manages state → Axios sends API requests.
- **API Gateway (Express.js):**
  Receives and routes requests to appropriate endpoints.
- **Backend (Node.js + Express):**
  - Auth middleware (passkey-based; JWT planned).
  - Role check for admin vs user.
  - Route → Controller → Service → Repository.
- **Database (MySQL via Sequelize):**
  Models define table structure. Repositories abstract all DB operations. Sequelize ORM acts as the bridge.
- **Frontend receives JSON response** and displays updated UI.

🔐 **Security:**

- **Current Security:** Passkey-based Admin registration.
- **Planned:**
  - JWT Authentication for route protection
  - Role-based Middleware for User/Admin separation
  - Password hashing with bcrypt

☑ **Benefits:**

- ✓ **Testability:** Each layer (controllers, services, repositories) is testable independently.
- ✓ **Maintainability:** Logic and data access are cleanly separated.
- ✓ **Scalability:** Easily extendable with modules like comments, notifications, analytics.
- ✓ **Flexibility:** Future upgrades (JWT, UI styling, database changes) can be integrated smoothly.

Process Flow Explanation:

1. Frontend Interaction:

- Users (Students, Professors, Supporting Staff, Admins) interact with the system through a React-based frontend.
- Actions like "Register", "Login", "Raise Complaint", "Like/Dislike", "Update Status" trigger HTTP requests.
- Based on the role and complaint type, different UI blocks (e.g., StudentRegister, NewComplaint, AdminComplaintFeed) are rendered dynamically.

2. **API Layer (Express.js):**

   - The frontend sends these requests to the Express.js API using Axios (e.g., /api/students/register, /api/publicComplaints).
   - While JWT is planned for future, currently a secure passkey system controls Admin access.
   - Middleware (basic or future auth.middleware.js) ensures:
     - Only admins can perform sensitive operations like update/delete complaints.
     - Users are limited to actions like submit, like, or view.

3. **Controller Layer:**

   - Each entity/module has a dedicated controller:

     students.controller.js, publicComplaints.controller.js, admins.controller.js, etc.

   - Controllers:
     - Handle incoming API requests
     - Validate request data
     - Forward logic to appropriate service functions

4. **Service Layer:**

   - Implements the core business logic.
   - Responsibilities:

     Validate and clean user input

     Apply complaint rules (e.g., expose = yes/no)

     Determine complaint visibility and access

   - Calls the repository layer to perform DB operations.

5. **Repository Layer:**

   - Abstracts all database access logic.
   - Each module has its own repository (e.g., students.repository.js, publicComplaints.repository.js).
   - Uses Sequelize ORM to perform actions like:
     - findAll, create, update, delete
   - Keeps data access logic separate from business rules, similar to JPA-style repositories in Java.

6. **Database Layer (MySQL via Sequelize ORM):**

   - All complaint and user data is stored in MySQL.
   - Tables include:
     - students, professors, supportingStaffs, publicComplaints, privateComplaints, admins, etc.
   - Sequelize:

- Auto-generates SQL tables from model definitions (e.g., students.model.js)
- andles relations, validations, and queries

Modules Implemented

Each module in the system follows the standard structure:
model.js → controller.js → service.js → repository.js

Core Modules:

Module Name	Description
Professors	Registration and login functionality for professor users.
Students	Handles student registration, login, and complaint access.
SupportingStaffs	Manages supporting staff registration and interaction with the system.
Admins	Admin registration via passkey, login, and complete complaint moderation.
PublicComplaints	Complaints submitted by users with name and optionally exposed to all.
PrivateComplaints	Anonymous complaints with optional exposure (visible only to admin or to all).

Each module includes:

- *.model.js – Defines the schema using Sequelize
- *.controller.js – Manages incoming API requests
- *.service.js – Implements validation and business rules
- *.repository.js – Handles database interactions

Entity Associations

Entity	Associations
Student	One-to-Many → PublicComplaints, PrivateComplaints
Professor	One-to-Many → PublicComplaints, PrivateComplaints
SupportingStaff	One-to-Many → PublicComplaints, PrivateComplaints
Admin	Can access and update all complaints
PublicComplaint	Many-to-One → User (Student/Professor/Staff) via userId
PrivateComplaint	Optional relation to user (can be anonymous)

Student Registration Flow Explanation (With Code References)

Frontend – Register.js

- When the user selects **"Student"** from the dropdown in Register.js, the student registration form is dynamically rendered with the necessary input fields of students.
- In StudentRegister.js, when the user submits the form, the following function is triggered:

```
await studentService.addStudent(details);
```

This calls the function in:
📁 **frontend/services/studentService.js**

Frontend Service – studentService.addStudent(details)

```
async addStudent(details) {
const response = await axios.post(`${STUDENT_API_URL}/`, details);
return response.data;
}
```

- API URL = http://localhost:5000/api/students/
- Method: **POST**
- Sends student form data to backend

Backend – Route: students.routes.js

```
router.post("/", async (req, res) => {
  const Student = await studentsService.addNewStudent(req.body);
  res.json(Student);
});
```

- This route receives the POST request and sends the data to the **service layer**.

Backend Service – students.service.js

```
async addNewStudent(studentDetail) {
  return await studentsRepo.create(studentDetail);
}
```

- Delegates the database creation to the **repository layer**

Backend Repository – students.repository.js

```
async create(studentDetails) {
  return await Students.create(studentDetails);
}
```

- This uses Sequelize to insert a new student into the **Students model/table**.

Model – students.model.js

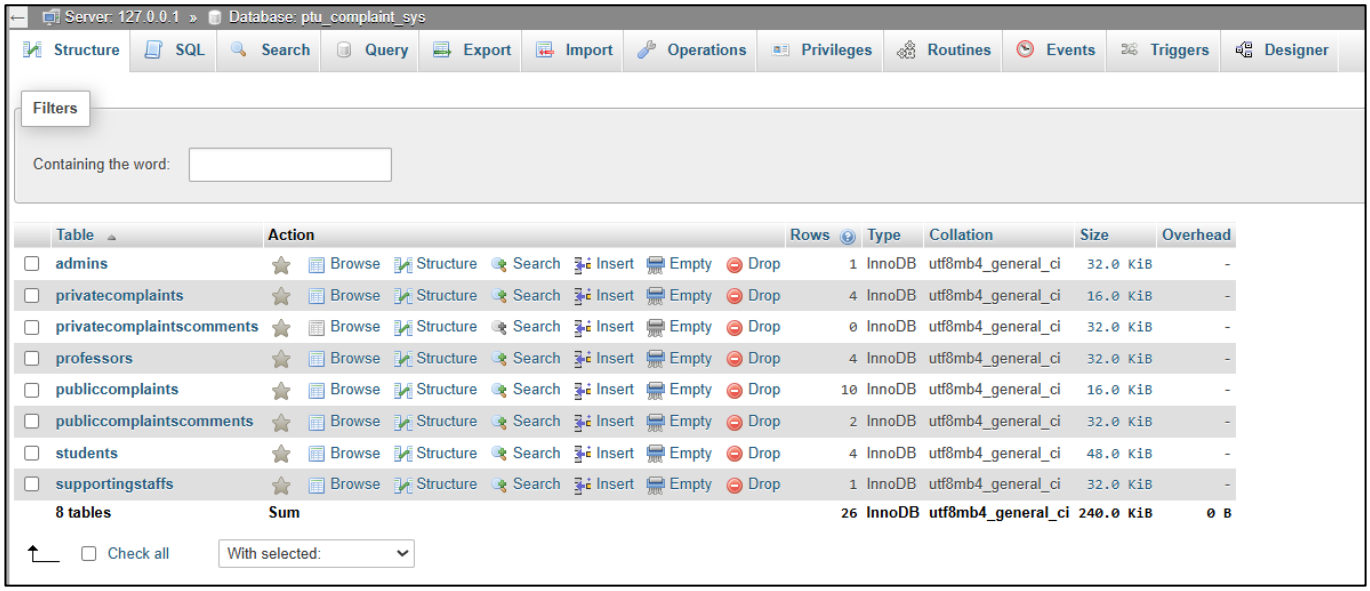
```
const Students = sequelize.define('Students', {
  id: { type: DataTypes.INTEGER, primaryKey: true, autoIncrement: true },
  registerNumber: { type: DataTypes.BIGINT, unique: true, allowNull: false },
  name: { type: DataTypes.STRING, allowNull: false },
  gender: { type: DataTypes.ENUM('male', 'female', 'other'), allowNull: false },
  email: { type: DataTypes.STRING, unique: true, allowNull: false },
  degree: { type: DataTypes.ENUM('Btech', 'Mtech', 'Phd'), allowNull: false },
  department: { type: DataTypes.ENUM('CSE', 'ECE', 'IT', ...), allowNull: false },
  batch: { type: DataTypes.STRING, allowNull: false },
  password: { type: DataTypes.STRING, allowNull: false }
});
```

- Defines the database schema for the student.
- Data is inserted into the **Students table in MySQL**.

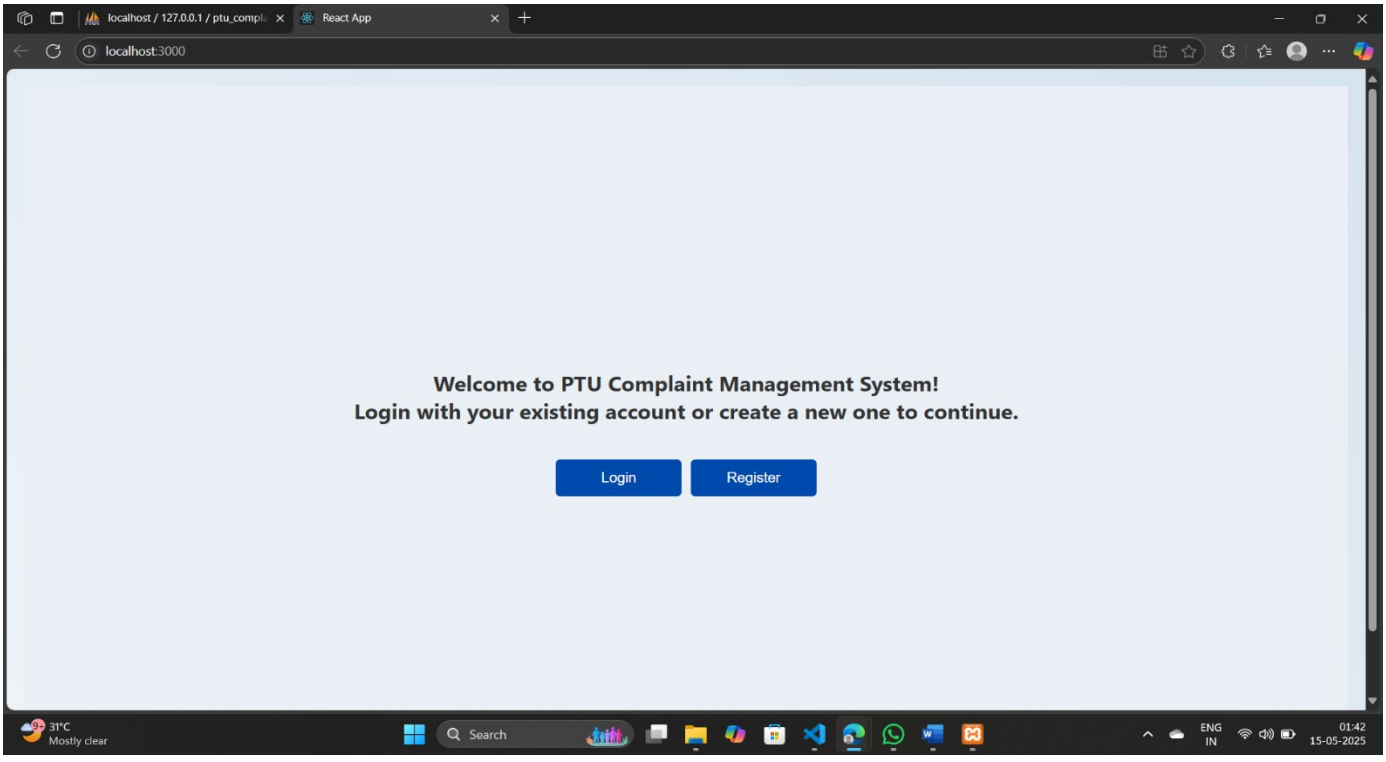
Final Result:

The student record is now stored in the MySQL DB, and the frontend receives a success response to proceed with login and redirect to Home page.

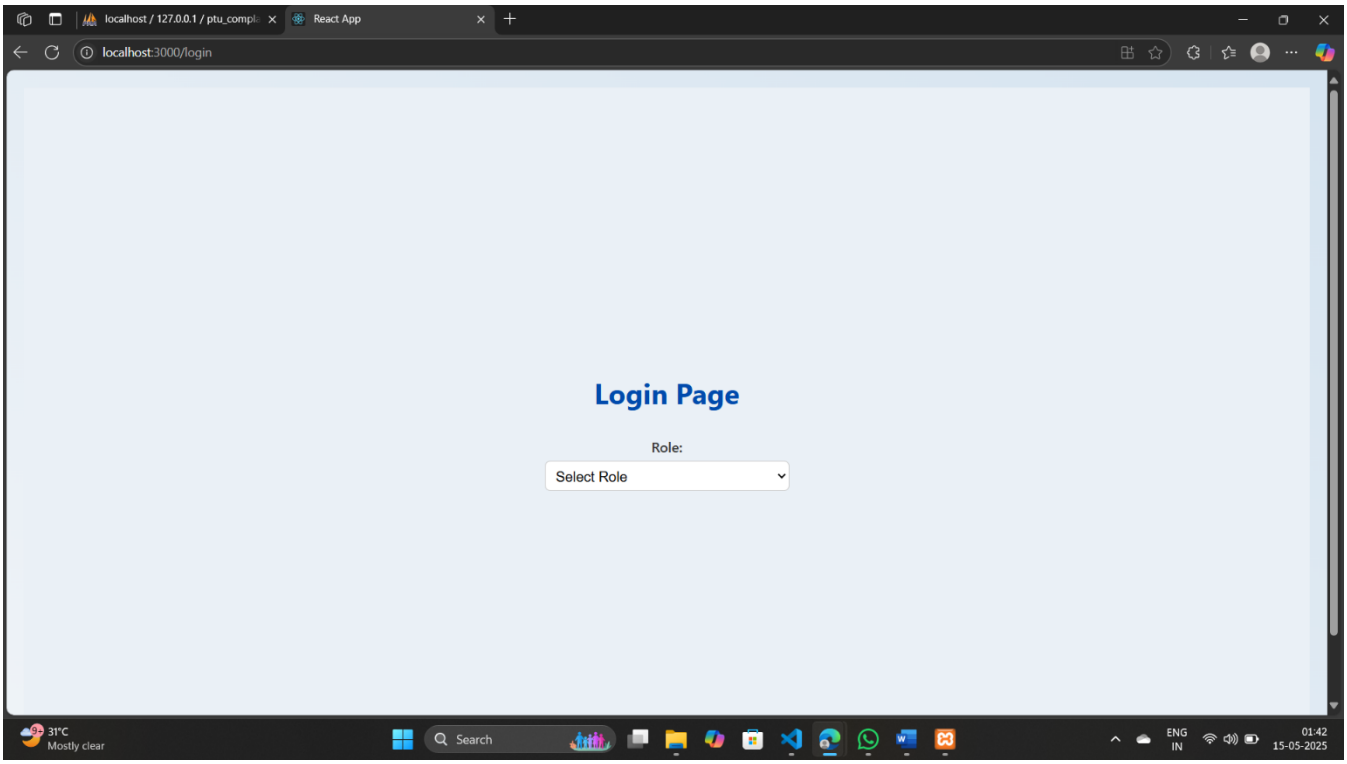
DATABASE



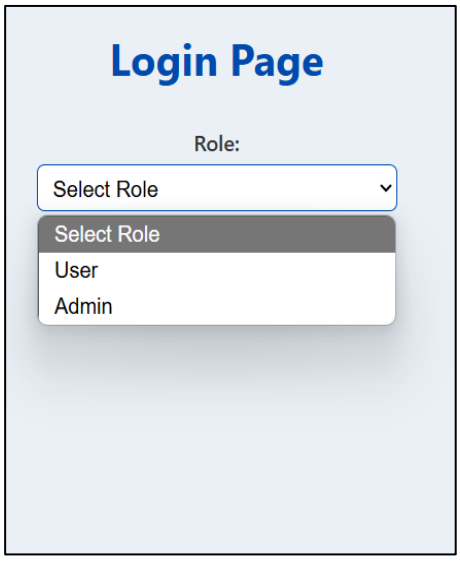
WELCOME PAGE

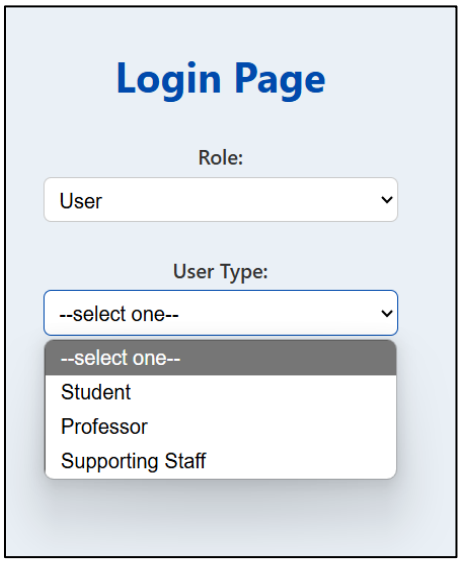


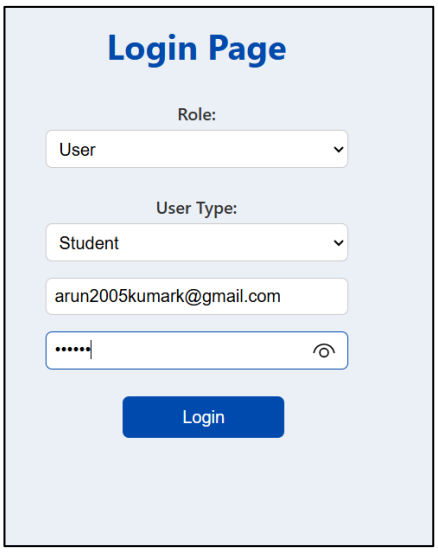
LOGIN PAGE



Login form dynamically rendered based on the selected Role(user/admin) and based on user type (student/professor/supporting staff)



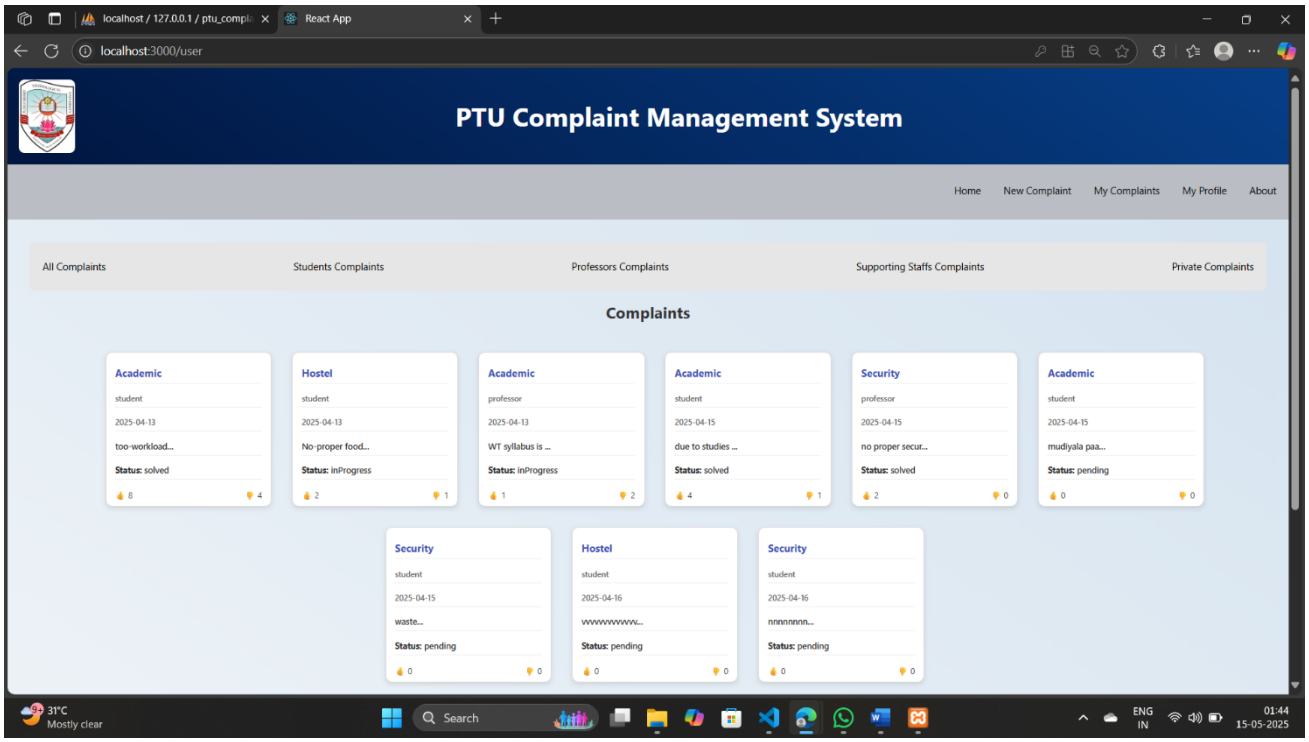




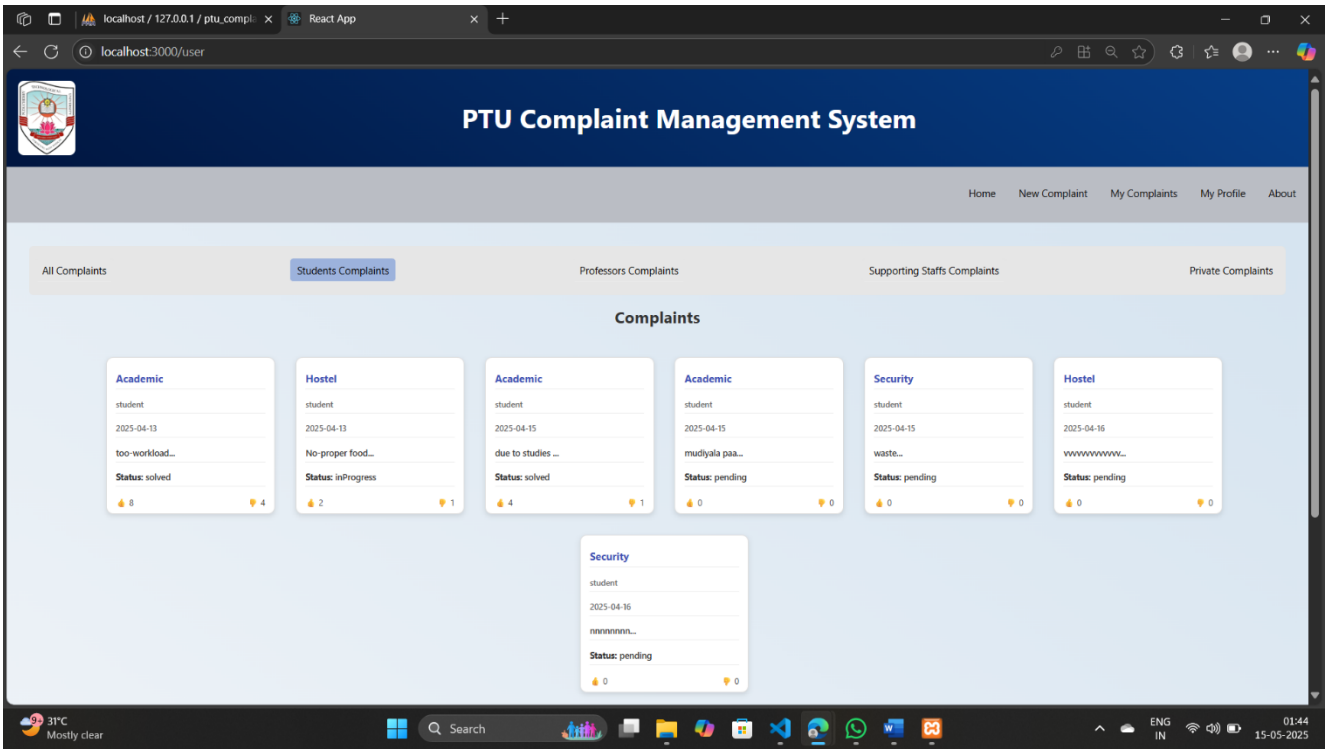
HOME PAGE

All the complaint cards are displayed and have a navbar to navigate to other pages, inner navbar to filter complaints based on the issuer type

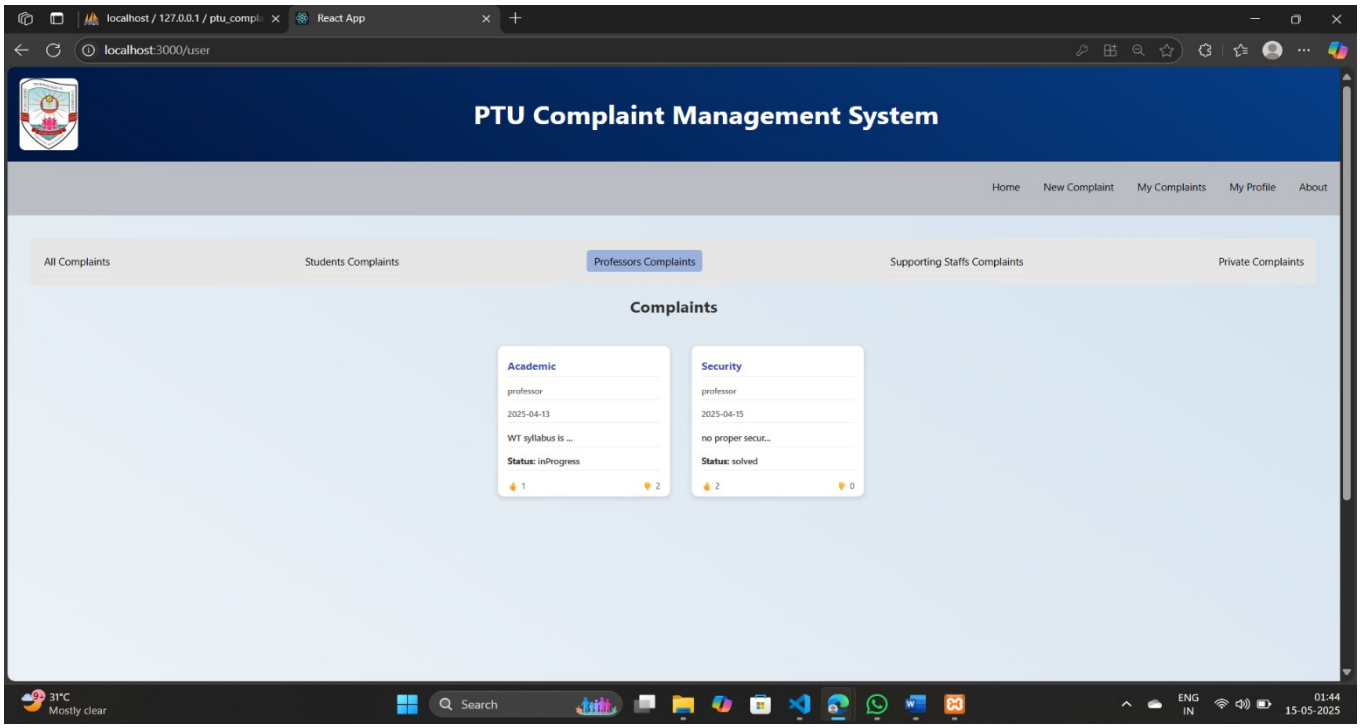
ALL COMPLAINTS



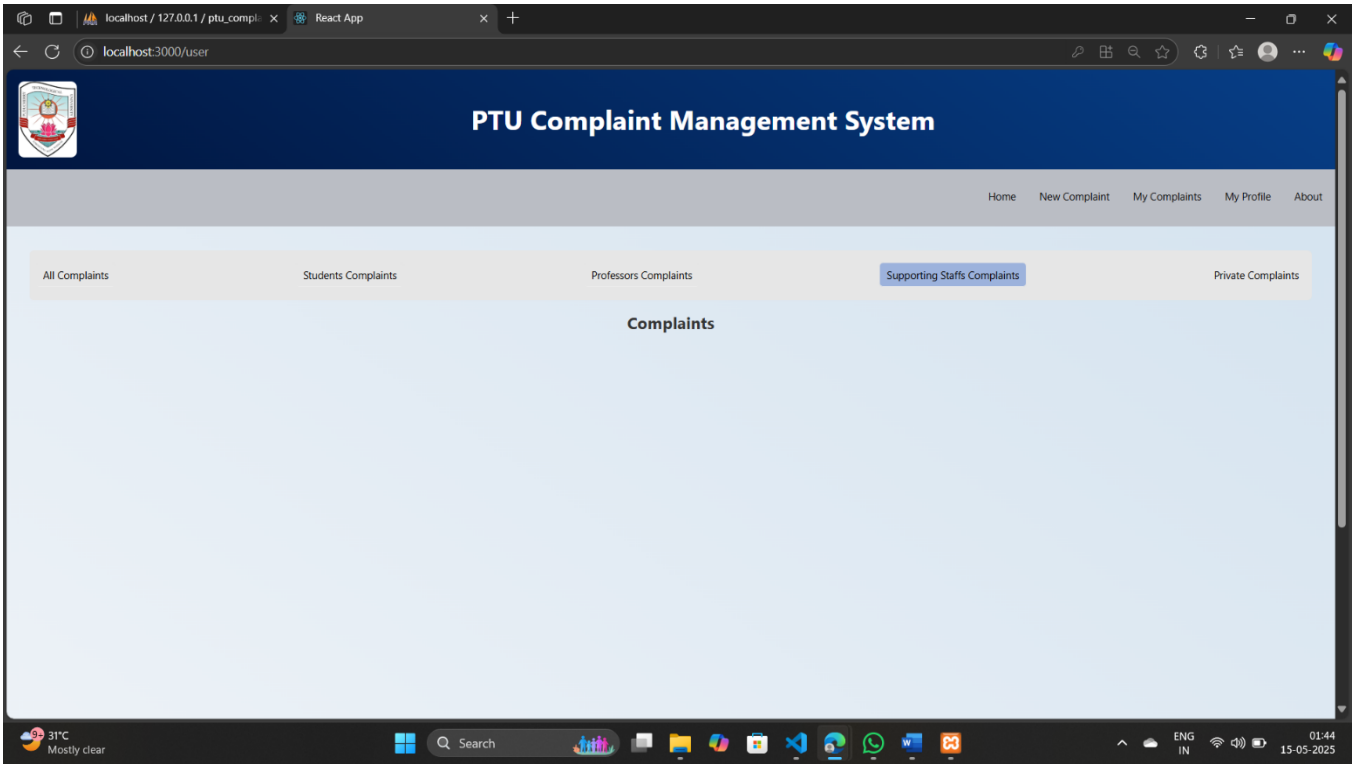
STUDENTS COMPLAINTS



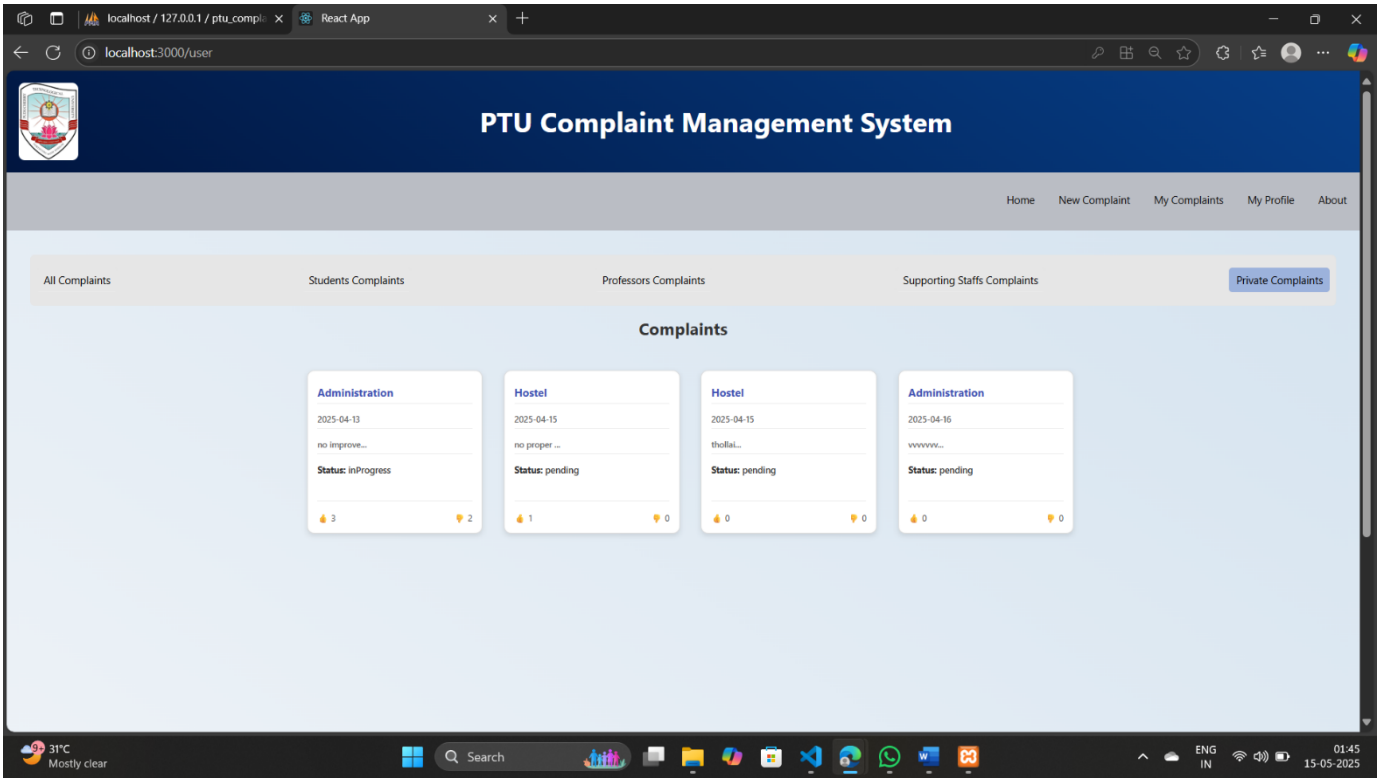
PROFESSORS COMPLAINTS



SUPPORTING STAFFS COMPLAINTS

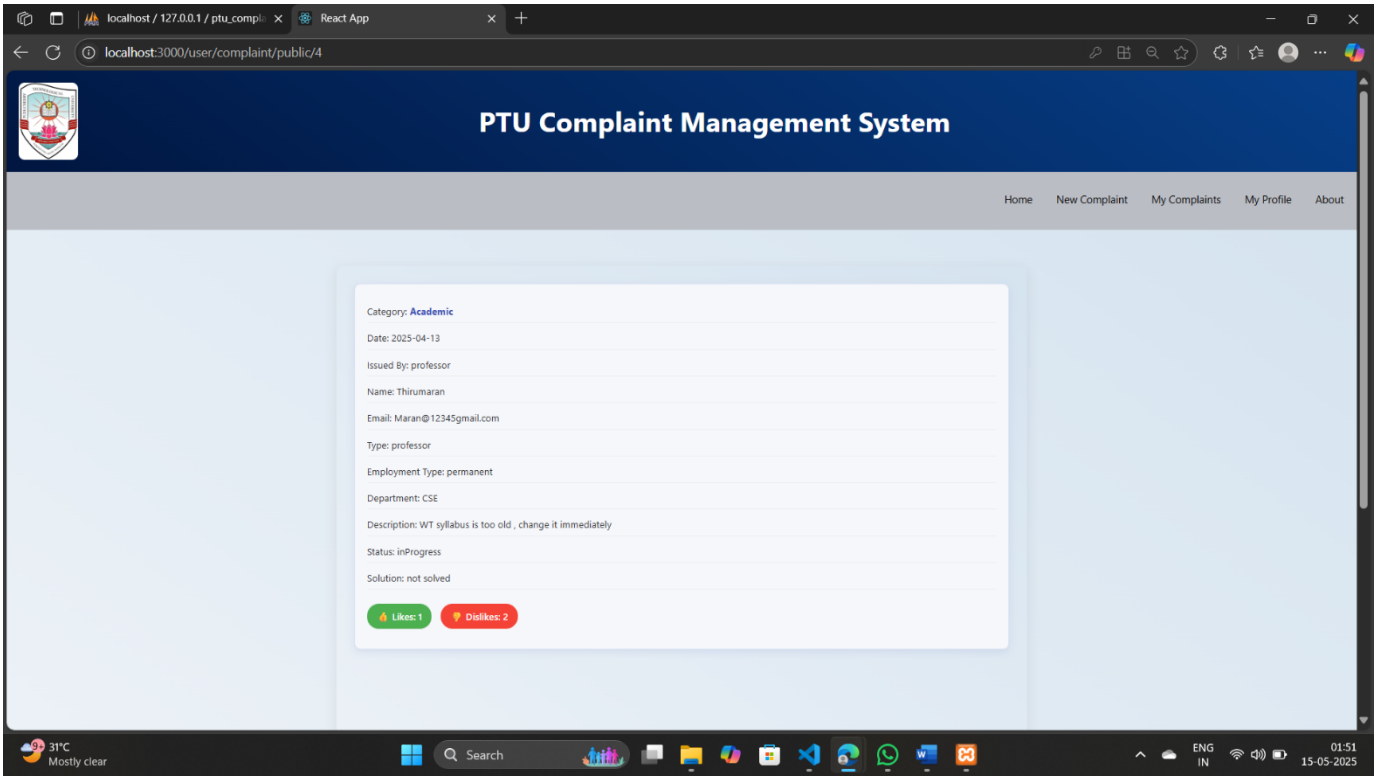


PRIVATE COMPLAINTS (without issuer details)

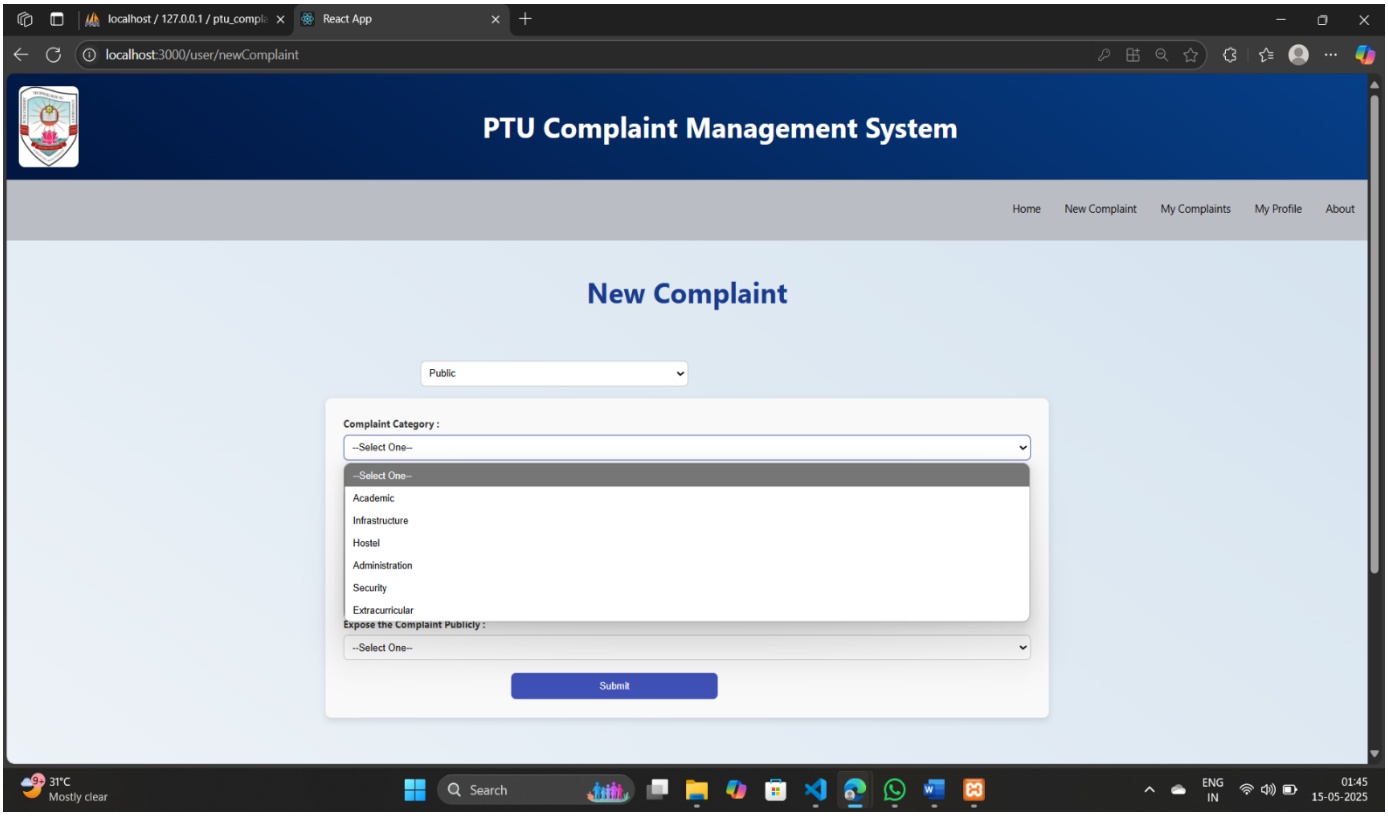


COMPLAINT VIEW

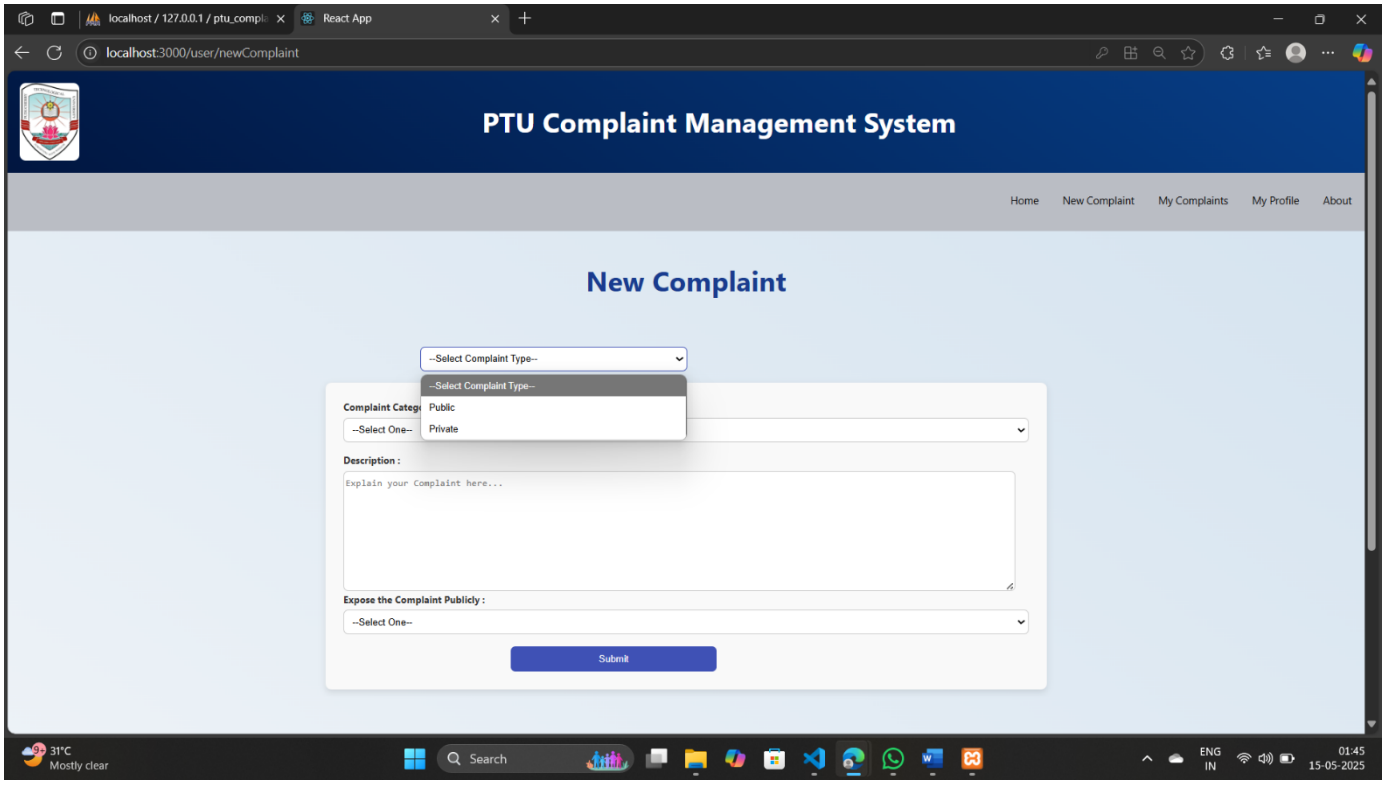
User can view the full complaint details by clicking the complaint card and can like and dislike the complaint to show their support to that complaint



NEW COMPLAINT ISSUING



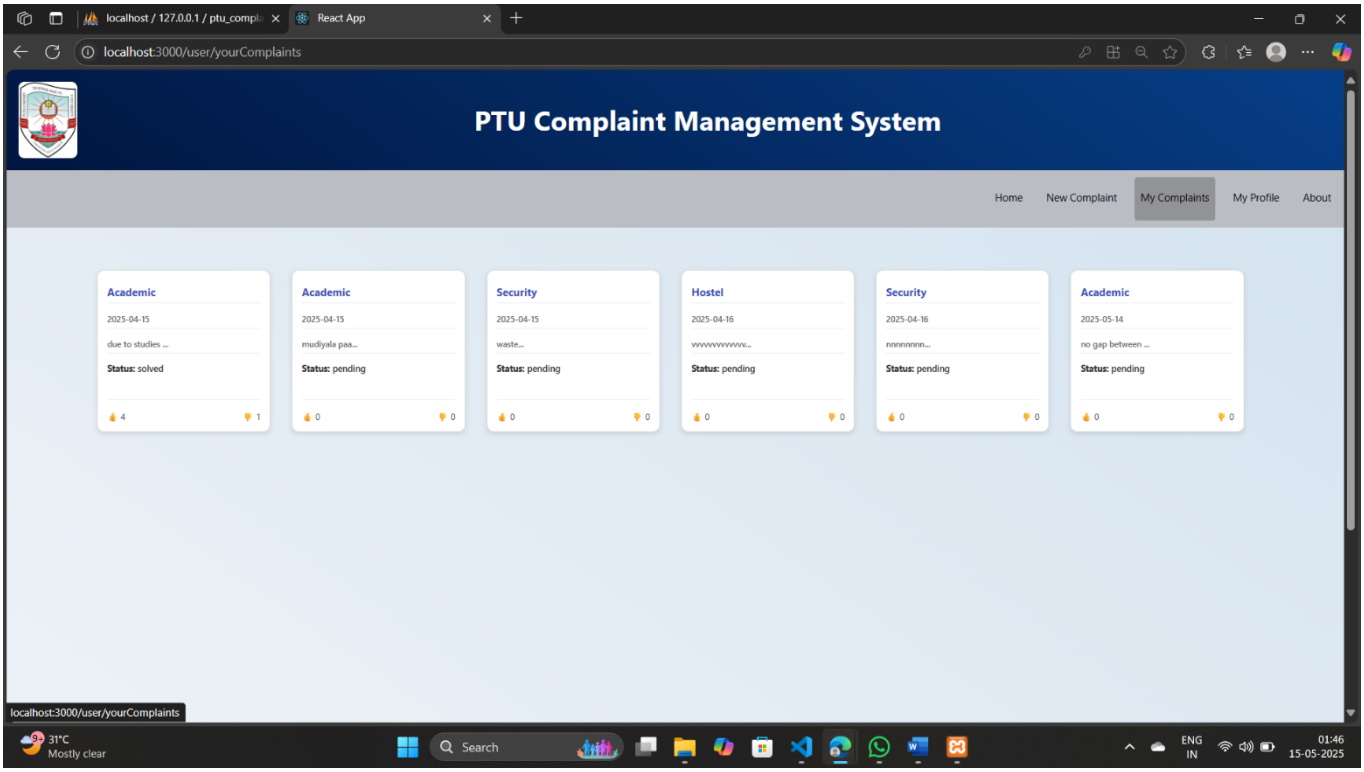
Complaint can be a type of public (with issuer details) / private (without issuer details)



If expose = ‘yes’ , complaint is visible to all in the website, if ‘No’ then only admin can see the complaint.

MY COMPLAINTS

User can see all their complaints, can check the status of the complaint.

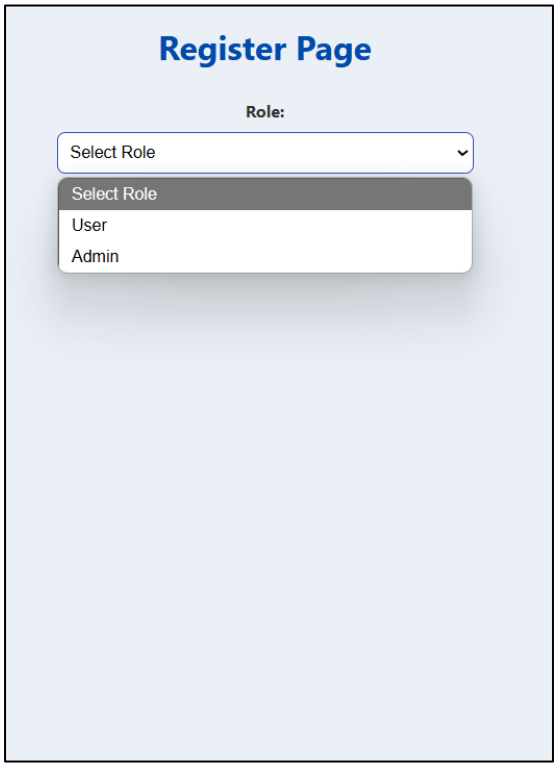


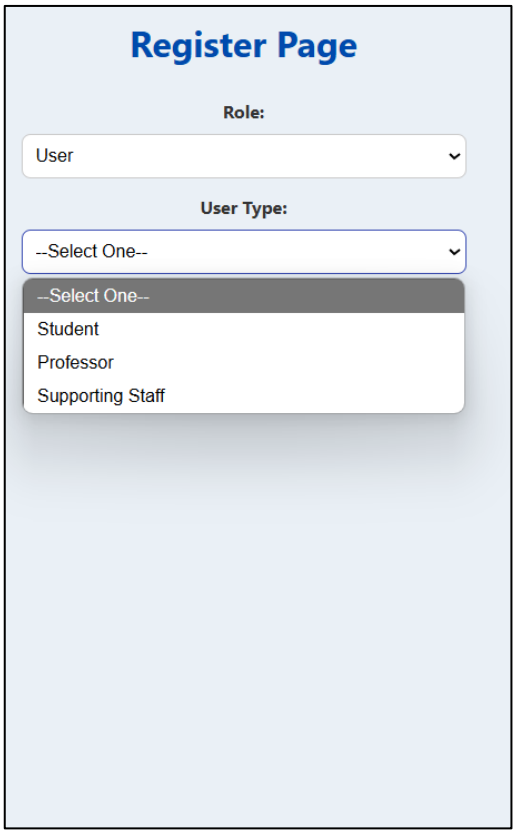
USER PROFILE PAGE

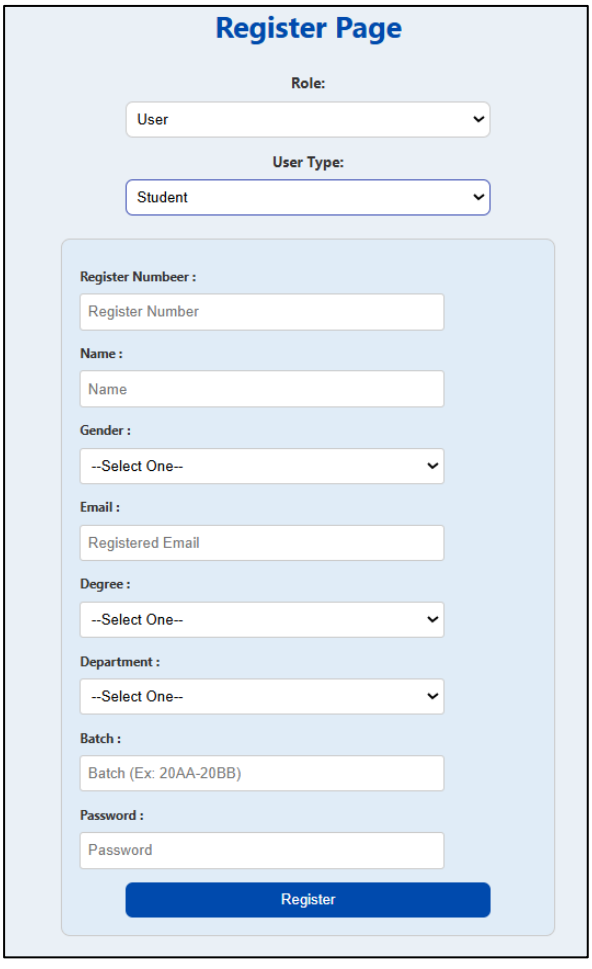
ABOUT PAGE

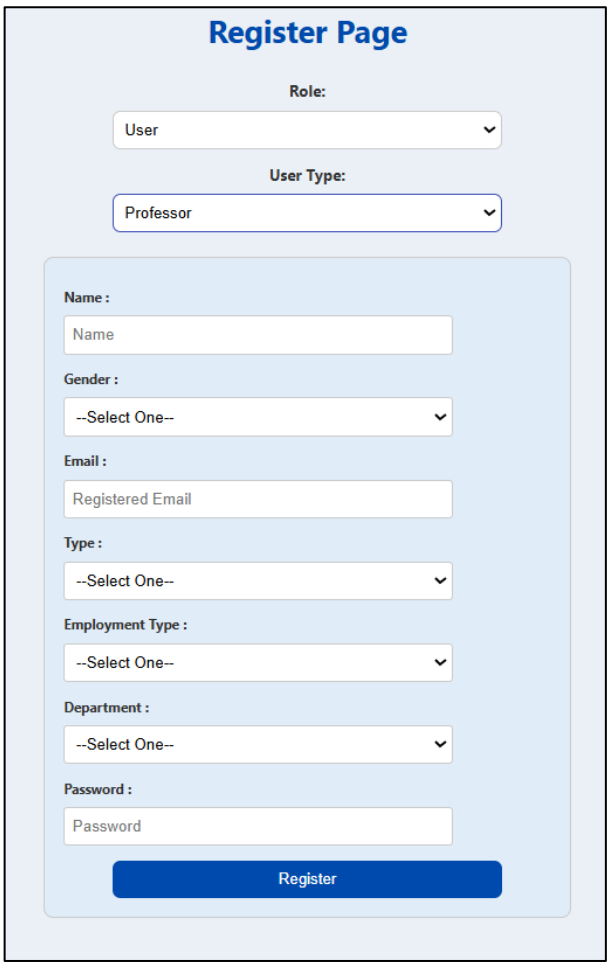
REGISTER PAGE

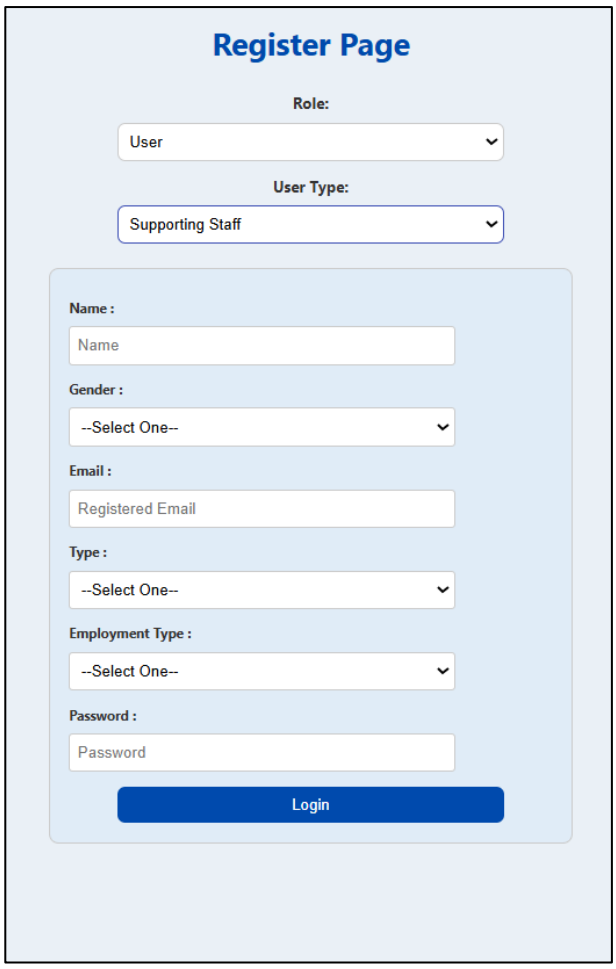
Register form dynamically rendered based on user role and type.

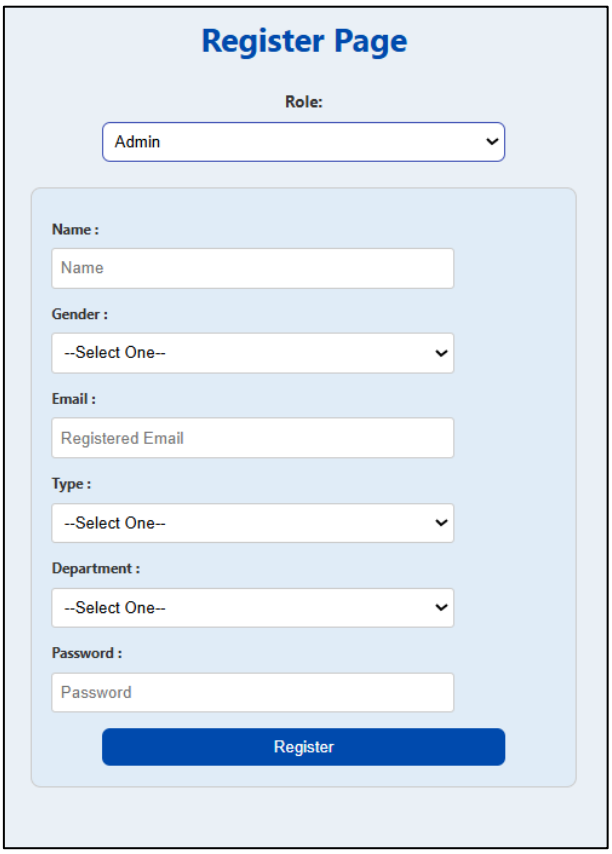




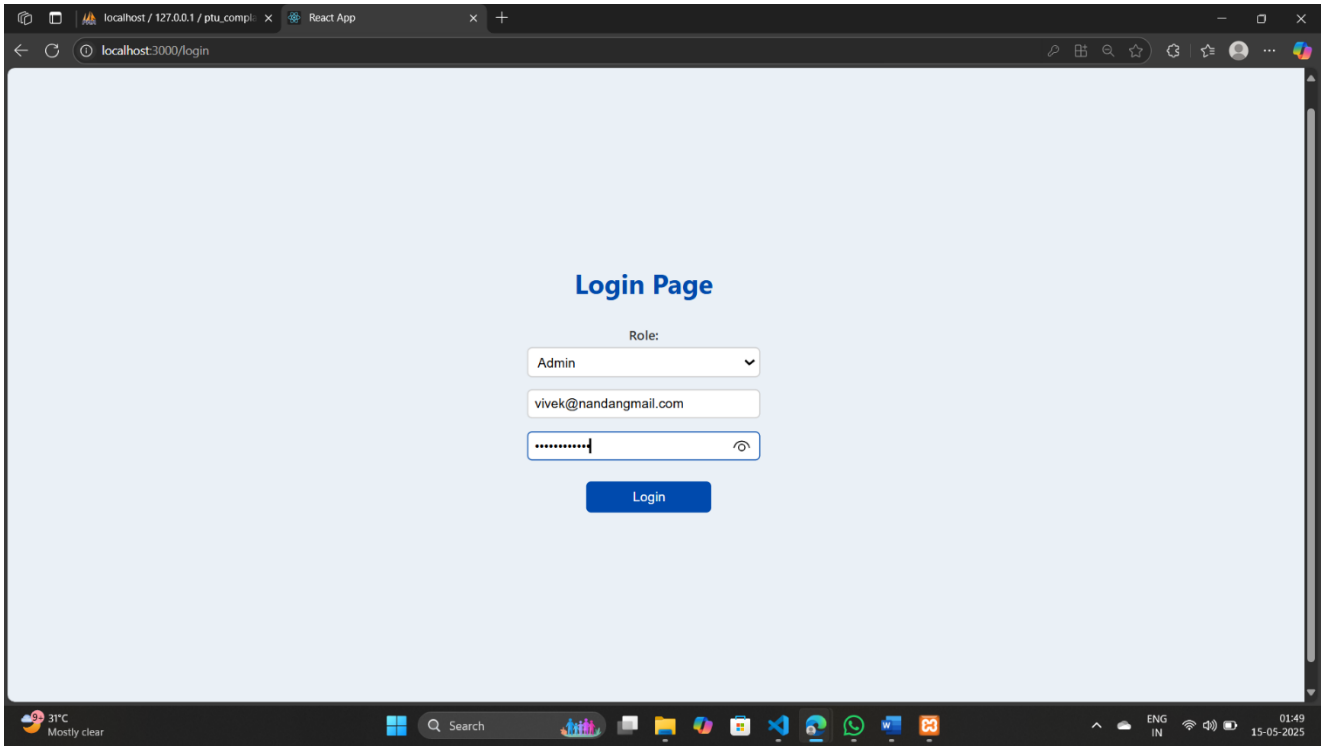





Register Page
Role:
User
User Type:
Supporting Staff
Name :
Name
Gender :
--Select One--
Email :
Registered Email
Type :
--Select One--
Employment Type :
--Select One--
Password :
Password
Login


Register Page
Role:
Admin
Name :
Name
Gender :
--Select One--
Email :
Registered Email
Type :
--Select One--
Department :
--Select One--
Password :
Password
Register

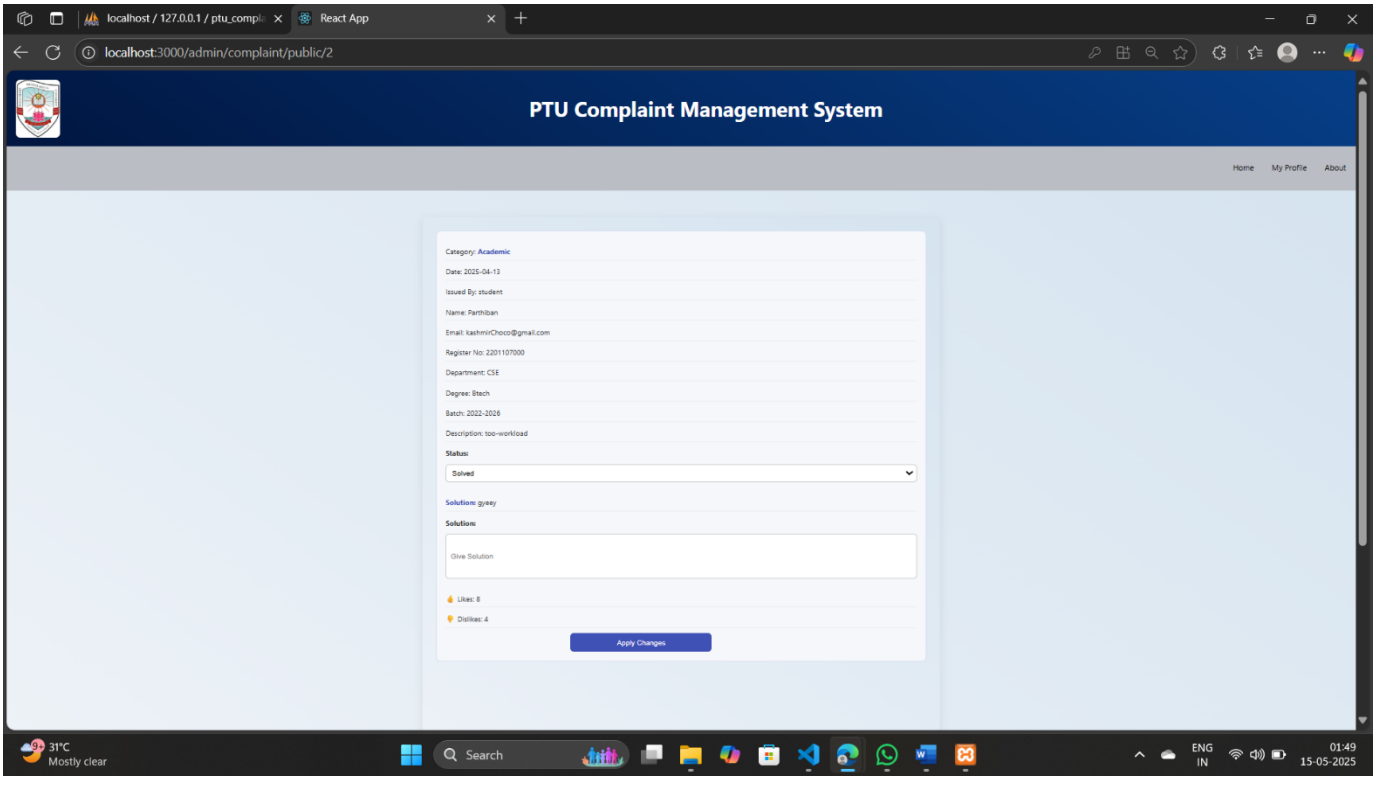
ADMIN LOGIN

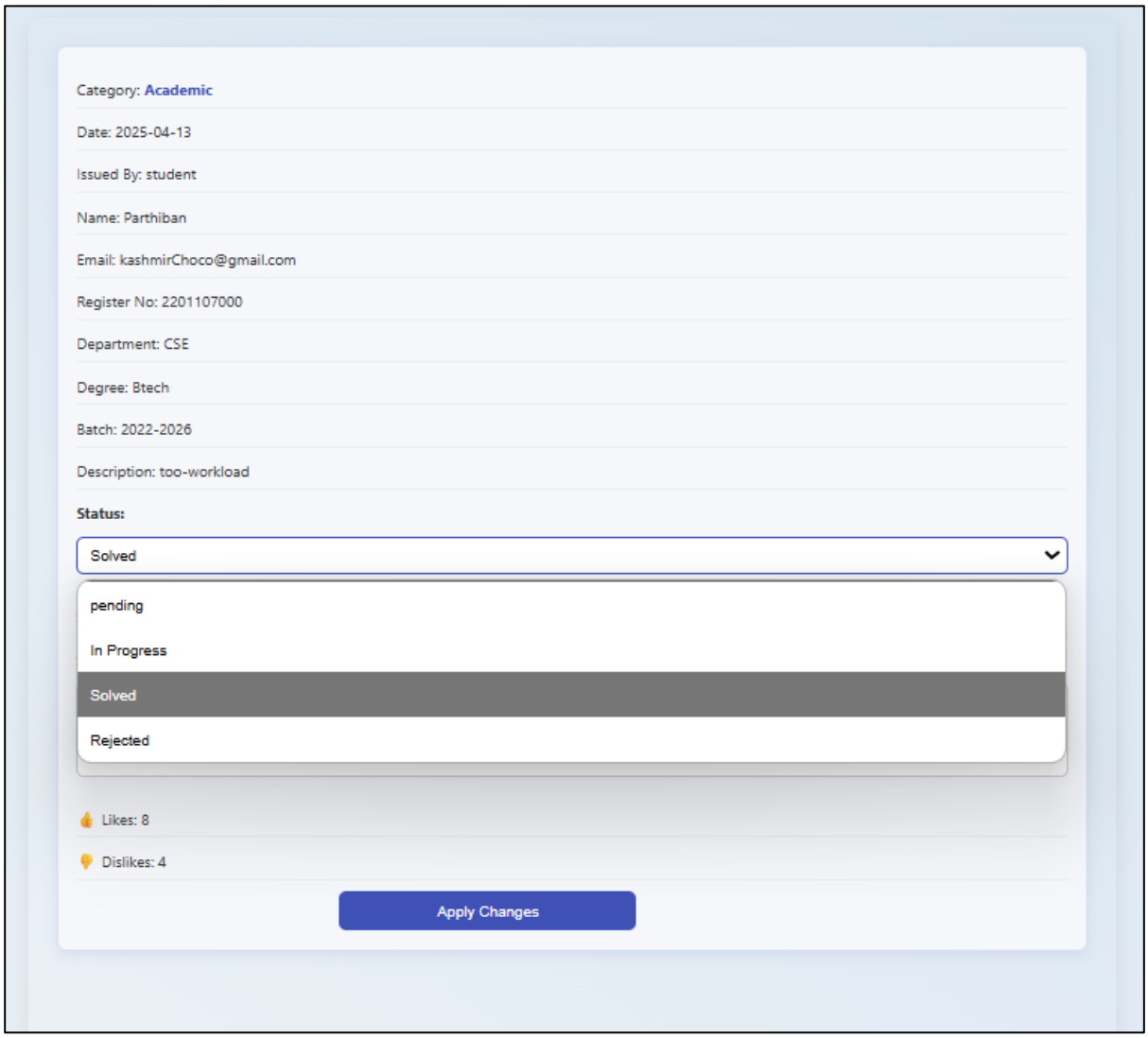

localhost / 127.0.0.1 / ptu_compl
React App
localhost:3000/login
Login Page
Role:
Admin
vivek@nandangmail.com
Login
31°C
Mostly clear
Search
ENG
IN
01:49
15-05-2025

ADMIN HOME PAGE

ADMIN – COMPLAINT VIEW

Admin can change the status and solution attribute of the complaint.





ADMIN PROFILE